Merging and mixing different languages: code wrapping and the coding glue

Antun Balaž
Scientific Computing Laboratory
Institute of Physics Belgrade
http://www.scl.rs/

27 Feb 2012

Advanced School on Scientific Software Development, ICTP, Trieste, 20 Feb – 2 Mar 2012

Overview

- Motivation
- Datatypes
- Arrays
- Linking
- Function arguments
- Common blocks
- Examples

Motivation

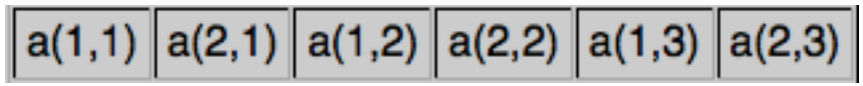
- Why we would want to mix different programming languages?
  - Numerical libraries
  - Code re-use
  - Collaboration
- We do it on a function level (subroutine)

Comparison of datatypes

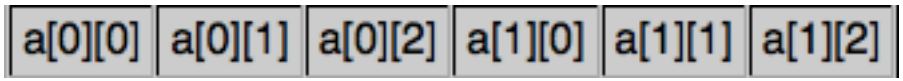
FORTRAN	C/C++
byte	unsigned char
integer*2	short int
integer	long int or int
integer iabc(2,3)	int iabc[3][2];
logical	long int or int
logical*1	bool (C++, One byte)
real	float
real*8	double
real*16	long double
complex	struct{float realnum; float imagnum;}
double complex	struct{double dr; double di;}
character*6 abc	char abc[6];
character*6 abc(4)	char abc[4][6];
parameter	#define *PARAMETER value*

Arrays

- Order of multi dimensional arrays in C/C++ is the opposite of FORTRAN.
- Native FORTRAN layout (column-major order): INTEGER A(2,3)



- Native C layout (row-major order) is **NOT** equivalent to the FORTRAN layout: int a[2][3];



Accessing arrays

- Equivalent multi-dimension arrays:
  - FORTRAN 77: INTEGER I(2,3,4)
  - FORTRAN 90: INTEGER, DIMENSION(2,3,4) :: I
  - C: int i[4][3][2];
- Native FORTRAN:

```
   INTEGER    I,J,NI,NJ
   PARAMETER(NI=2,NJ=3)
   INTEGER    B(NI,NJ)                    same as B(2,3)

   DO J = 1, NJ                           loop to 3
       DO I = 1, NI                       loop to 2
           PRINT 10, I, J, B(I,J)
10         FORMAT('B(', I1, ',', I1, ') =', I2)
       END DO
   END DO
```

- Native C layout:

a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]

```
int a[3][2])
int i, j;

for (i = 0; i < 3; i++)
    for (j = 0; j < 2; j++)
        printf("a[%d][%d] = %d\n", i, j, a[i][j]);
```

Linking

- Fortran subroutines are the equivalent of "C" functions returning "(void)".
- The entry point names for some FORTRAN compilers have an underscore appended to the name. This is also true for common block/structure names:

FORTRAN	C
CALL SUBRA(...)	subra_(...)

- The f77 compiler flags "-fno-underscore" and "-fno-second-underscore" will alter the default naming in the object code and thus affect linking
- The case in FORTRAN is NOT preserved and is represented in lower case in the object file. The g77 compiler option "-fsource-case-lower" is default. GNU g77 FORTRAN can be case sensitive with the compile option "-fsource-case-preserve”

Profiling Fortran linked with C

- When debugging with GDB, the Fortran subroutines must be referenced with names as they appear in the symbol table. This is the same as the C representation
- Thus when setting a break point at the Fortran subroutine subra(), issue the comand "break subra_".

Function arguments (1)

- All arguments in FORTRAN are passed by reference and not by value, so C must pass pointers

FORTRAN	C
call subra(i, x)	subra_(int *i, float *x)

- Characters in Linux and GNU compilers: When passing character strings, the length must follow as separate arguments which are passed by value

FORTRAN	C

call subra(string_a, string_b)	len_a = strlen(string_a);
	len_b = strlen(string_a);
	subra_(char *string_a, len_a, char *string_b, len_b)

- On some Unix compilers: When passing character strings, the length must be appended as separate arguments which are passed by value.

FORTRAN	C
call subra(string_a, string_b)	subra_(char *string_a, char *string_b, len_a, len_b)

SCIENTIFIC COMPUTING LABORATORY

Function arguments (2)

- C expects strings to be null-terminated. Thus character strings from FORTRAN which are passed back to C should be null terminated with CHAR(0)

```
CHARACTER(LEN=32) :: sample_string = "This is a sample"//CHAR(0)

or

CALL SUBRA ('A string'//CHAR(0))
```

Common blocks

- Fortran common block and global C/C++ extern structs of same name are equivalent. Never use un-named common blocks!
- Reference variables in same order, same type and with the same name for both C and FORTRAN. Character data is aligned on word boundaries.

```
FORTRAN:
      DOUBLE PRECISION X
      INTEGER A, B, C
      COMMON/ABC/ X, A, B, C

C:
      extern struct{
          double x;
          int a, b, c;
      } abc_;

C++:
    extern "C" {
      extern struct{
          double x;
          int a, b, c;
      } abc_;
    }
```

- Use of extern requires that the common block be referenced first by FORTRAN. If referenced first by C then drop the extern. The extern statement states that it is trying to reference memory which has already been set aside elsewhere.

Example: Calling C (1)

testF.f

```
      program test

      integer ii, jj, kk
      common/ijk/ ii, jj, kk
      real*8  ff
      character*32 cc

      ii = 2
      jj = 3
      kk = 4
      ff = 9.0567
      cc = 'Example of a character string'

      write(6,10) ii, ff
10    format('ii= ',i2,' ff= ',f10.4)

      call abc(ii)

      write(6,20) ii
20    format('ii= ',i2)
```

SCIENTIFIC COMPUTING LABORATORY

Example: Calling C (2)

```
      write(6,30) ii, jj, kk

      call doubleIJK(cc)

      write(6,30) ii, jj, kk
30    format('ii= ',i2,' jj= ', i2, ' kk= ', i2)

      write(6, 40) cc
40    format(a32)

      stop
      end

      subroutine abc(jj)
      jj = jj * 2
      return
      end
```

SCIENTIFIC COMPUTING LABORATORY

Example: Calling C (3)

testC.c

```
#include <stdio.h>

extern struct
{
   int ii, jj, kk;
} ijk_;

int doubleijk_(char *cc, int ll)
{
   cc[ll--] = '\0';  // NULL terminate the string

   printf("From doubleIJK: %s\n",cc);

   ijk_.ii *=2;
   ijk_.jj *=2;
   ijk_.kk *=2;

   return(1);
}
```

SCIENTIFIC COMPUTING LABORATORY

Example: Calling C (4)

- f77 -c testF.f
- gcc -c testC.c
- f77 -o test testF.o testC.o
- If you use C/C++ standard libraries, you may have to include linking arguments -lc or -lstdc++

Example: Calling Fortran (1)

testC.cpp

```
#include <iostream>

using namespace std;

extern"C" {
void fortfunc_(int *ii, float *ff);
}

main()
{

   int ii=5;
   float ff=5.5;

   fortfunc_(&ii, &ff);

   return 0;
}
```

Example: Calling Fortran (2)

testF.f

```
      subroutine fortfunc(ii,ff)
      integer ii
      real*4  ff

      write(6,100) ii, ff
100   format('ii=',i2,' ff=',f6.3)

      return
      end
```

- f77 -c testF.f
- g++ -c testC.cpp
- g++ -o test testF.o testC.o -lg2c
- When mixing Fortran with C++, name mangling must be prevented.
-

References

- http://arnholm.org/software/cppf77/cppf77.htm
- http://www.yolinux.com/TUTORIALS/LinuxTutorialMixingFortranAndC.html